# Controlling a programming environment through a voice based virtual assistant

**Sonia Grigor**
Technical University of Cluj-Napoca, Computer-Science Department
Cluj-Napoca, Romania
sonia.grigor@student.utcluj.ro

**Constantin Nandra**
Technical University of Cluj-Napoca, Computer-Science Department
Cluj-Napoca, Romania
constantin.nandra@cs.utcluj.ro

**Dorian Gorgan**
Technical University of Cluj-Napoca, Computer-Science Department
Cluj-Napoca, Romania
dorian.gorgan@cs.utcluj.ro

DOI: 10.37789/rochi.2020.1.1.18

**ABSTRACT**

**The use of smart personal assistants is intended to provide a solution that employs the voice interaction model in order to help improve accessibility for everyday tasks. This model is more than suitable for simple tasks, such as internet searches or device-controlling commands. In this paper we explore the possibility of using this interaction model for completing more complex, composite, context-dependent tasks. Particularly, we look into the potential benefits of using custom spoken commands to help novice users develop insight into the workings of a computer program. Throughout this paper, we present a solution based on an existing, customizable voice assistant that is meant to both help users grasp the structure of a program and improve accessibility for programming activities. The latter is achieved by providing a framework for a voice-based programming environment, offering features like code fragment insertion, navigation, error detection, handling and program running, while also providing voice and text-based feedback for the executed commands.**

**Author Keywords**

Voice-based interaction; Programming assistant; Amazon Web Services; Echo Dot; Alexa;

**ACM Classification Keywords**

H.5.2. Information interfaces and presentation (e.g., HCI): User Interfaces. H.3.2. Information Storage and Retrieval: Information Storage.

**General Terms**

Human Factors; Design.

## INTRODUCTION

During the last decades, solutions developed by the IT (Information Technology) industry have become ever more prevalent within every kind of human activity, be it personal, social or economic. Computer programming is at the core of this industry and, because of this, software developers are required in order to create the applications impacting our lives. However, computer programming is one of the more challenging occupations, since it normally requires multiple levels of specialized education to adequately master.

The ability to program, in some form or another, is one of the most sought-after skills at workplaces around the world. This high demand for software developers can incentivize many individuals to follow such a carrier path. Nevertheless, acquiring and developing programming skills can be difficult for users with experience in areas other than software development, who lack the basics of a formal education and training. Possible ways to alleviate this problem might include the training of candidates through exposure to practical use-cases and basic solution implementations. This should be done in an intuitive manner, using simplified terminology and real-world, applied examples, while minimizing the use of high-level, abstract concepts.

Recent development in human-computer interaction have led to the rise of intelligent personal assistants. Being controlled through natural language, they offer a human-centered interaction model that is intuitive and easy to use, with little to no training required on the user's part.

The project presented within this paper intends to capitalize on the intuitive nature of the language based interaction model in order to facilitate the training of novice programmers. The idea is to employ an intelligent voice assistant and integrate it with a programming environment, thus allowing for voice commands to be used in controlling code insertion, editing and execution. The proposed solution would employ an intuitive, top-down approach, starting from the general structure of a program, and gradually working towards more specialized constructs and instructions.

The system provides a small and well-defined set of voice commands, with which the user would be able to learn to employ the most important syntax elements of a given programming language. The interaction between the user and the system would be performed on several channels. The system receives input from the user in the form of a spoken command, and then it provides two types of feedback: voice feedback through the response that notifies the user of the status of the executed command, and visual/text based feedback consisting of state changes within the programming environment and the supplying of information or error messages.

## RELATED WORK

Human-computer interaction has been studied since the advent of user interfaces and is closely related to the term of usability. This specifies the degree of satisfaction of a user with regard to his interaction with an application through a dedicated interface. In the past, the term computer-human interaction, placing emphasis on the computer, was used to describe the relationship between the two. Nowadays, with the evolution of technology and the attempt to reduce the time and effort required to effectively use a system, the emphasis is on the human element, with the term human-computer interaction accurately describing the current trend [1]. This is most obvious with the recent advent of handheld devices featuring personal assistant software. These are meant to facilitate the interaction of the human with the computer, exploiting increasingly powerful devices to mimic the natural human communication capabilities. Relevant examples include complex processing tasks, such as image processing and classification, text-to-speech and voice recognition.

### Virtual assistants

Voice assistants are software agents developed to intercept the human voice, interpret it and respond in a synchronous manner. Each of the biggest players on the IT market have developed one or more intelligent personal assistants to be delivered in different forms and with different roles to fulfill:

- embedded in the phone: *Siri*, *Google Assistant*;
- operating system functions: *Cortana*;
- dedicated devices:
  - *Alexa* embedded in smart speaker Echo devices
  - *Google Assistant* as part of *Google Home* home-automation devices [2].

#### *Alexa*

Today, the concept of an intelligent personal assistant is often associated with a smart speaker that can be activated by voice interaction using a wake word, and can perform a variety of user queries [3]. This is the case of *Alexa*, the virtual assistant developed by *Amazon*. It responds to the activation word "Alexa", and offers a sizeable set of standard commands, as well as the possibility for user customization. *Alexa* is the name of the voice service that powers *Amazon Echo* (the intelligent speaker), offering features or abilities that allow customers to interact with devices in a more intuitive, voice-based manner. The provided functionality includes, but is not limited to: internet searches, media playback and device control commands.

Figure 1 shows the method of user interaction with *Amazon Alexa*. First, users produce an utterance or request, which is filtered by *Alexa* through speech recognition, machine learning, and natural language processing. All of these are complex processes, requiring significant computational power and are therefore performed in the Cloud. Alexa then accesses web-hosted services, which employ this functionality, and provides a response to the user. Included in the response process, Alexa produces an information "Card" that records users' words and the resulting system response. The "Card" information is available to users via the *Alexa* application in a textual form, providing a record or interaction history [4].


**Figure 1. Alexa interaction model [4]**

#### *Google Assistant*

The virtual assistant provided by *Google* is available on smartphones and smart home devices. Its activation words are "Hey Google", or "OK Google" [5]. When the software is activated by voice, the user usually receives feedback in the form of a changed screen if it is a phone, or in the form of a light on the device, if it is a *Google Home* device. On mobile devices it is employed for device control purposes and user queries. Hands free phone operation and internet searches are among the most well-known use cases. In the case of *Google Home*, the assistant is most often employed for device control purposes, managing various types of home appliances [6].

#### *Siri*

The *Siri* virtual assistant developed by *Apple* is available on all *Apple* devices including *iPhones*, *MacBooks*, *iPads* and *Apple Watches* [7]. It responds to the activation word "Hey Siri" and uses voice queries and a natural language user interface to answer questions, make recommendations, and perform actions by delegating requests to a set of Internet services. Its voice recognition engine was provided by *Nuance Communications*, and uses advanced machine learning technologies to operate [8].

The ways in which *Siri* can be used are diverse and numerous. From common features such as initiating a call, sending a message, to unit conversions, and displaying notifications based on a user device's location, *Siri* finds applications in many areas.

Some specialized commands that *Siri* knows to execute include: adding a post on *Twitter* or *Facebook*, searching for a person's tweets, solving mathematical operations and setting alarms based on a given location, which is activated when the respective location is reached, the conversion of the units of measurement [9].

#### *Cortana*

*Cortana* is the personal assistant developed by *Microsoft* that helps users save time and focus on relevant aspects of utilizing an operating system. It is notable in its ability to

gather data on user activity and preferences in order to improve user interaction. Thus, coupled with regular updates to the operating system being pushed by *Microsoft*, give it the potential to continually improve. Some of *Cortana*'s functions include: calendar management, joining a meeting using *Microsoft Teams*, setting alarms, opening applications on the user's computer, help with system's management and settings and more [10].

### Similar applications

Interactive programming learning methods are either text-based or in the form of video tutorials available online. They provide potential students with the theoretical information and the opportunity to apply said information.

The authors of [11] developed an application to help beginners learn SQL. They developed both a graphical interface of the application, and provided a natural language processing engine. The paper presents the *Cyrus* application which has two main modes. Firstly, it acts as a guide, allowing students to choose a database on which to experiment with different queries, which can be specified vocally. Secondly, in its knowledge assessment mode, the system allows its users to filter questions based on difficulty levels and answer them through a text-based interaction model. While employed in tutorial mode, the system accepts the voice query in English, maps the query to SQL and executes it to produce the result. As a matter of redundancy, students can also write or edit the SQL themselves, bypassing the voice command interface.

*Voice Coder* [12] is an extension to *Alexa's* rule set that helps users create games using through voice commands. The game begins with no rules or logic. The user's main responsibility is to program rules, using events, activities, and values. Example of activities include moving or playing a sound.

*Coder* [13] is another example of an *Alexa* extension. It aims to teach its users programming by providing coding examples. It has support for more than 10 languages (some of them still in progress). Users can ask for examples and instructions on how to write code in some of the most popular languages.

*C Programming Quiz* [14] is a quiz based on questions about the C programming language. It has instructions for navigating between questions and it was created using available templates provided by *Alexa Skill Kit*.

*CS Guru* [15] provides users with a selection of questions from the Data Structures and Algorithms field, and has a weekly updated content. Question are straight-forward and they have answers and explanations that can be easily understood and reproduced. This application provides a training mode which is designed to teach users about the key concepts in Computer Science.

## SYSTEM OVERVIEW

The project presented in this paper aims to demonstrate how voice interaction with intelligent personal assistants can improve a tool's accessibility by simplifying the way in which certain tasks can be performed. In this context, the term "intelligent assistants" refers to specialized software using a knowledge base to process voice commands by employing sophisticated methods and algorithms for speech recognition and interpretation.

The project described within this paper explores the possibility of using voice interactions to help users learn the basics of programming through natural language. It also aims to provide the basis for a programming environment featuring improved accessibility, which might cater to users suffering from various motor disabilities. The system supports four different programming languages at the moment, with mechanisms in place to allow for easy extension. It permits any user, without knowledge of the system's workings to upload language elements structured in a specific manner, as well as predefined error messages for various situations which might appear when compiling the code.

The objectives of the application are linked to the needs of novice users in the field of programming. The application can meet several needs of such a user, related to learning about: the structure of a program, control structures (conditioned, repetitive or iterative), sorting algorithms like BubbleSort, MergeSort, and the logic of a program.

The overall architecture of the system is shown in Figure 2.

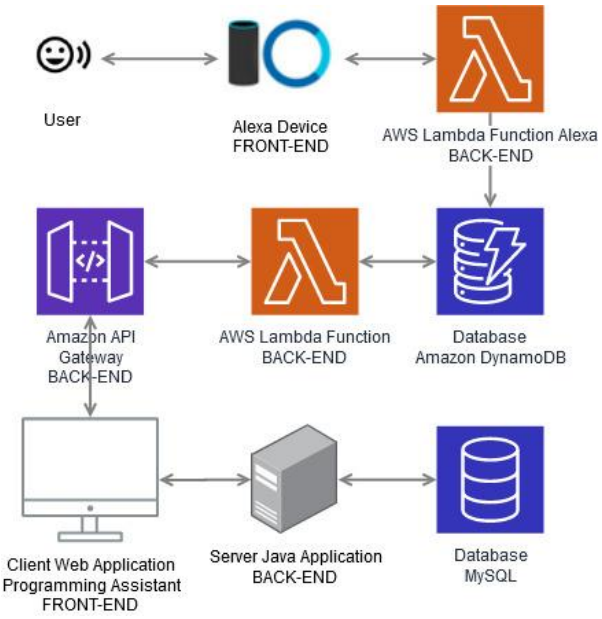


**Figure 2. System conceptual architecture**

The user interacts with the device hosting Alexa. This is called Amazon Echo, and represents a line of smart speakers

sold by Amazon. They can be controlled by voice and include the *Alexa* virtual assistant. The custom command interpretation logic of the assistant resides within a repository of *AWS Lambda Functions*, hosted remotely within *Amazon*'s Cloud. This is accessed by *Alexa* via exposed web services every time that it needs to process a custom command. This logic is pretty low level, is implemented by the user and (in our case) consists in generating a specified code and a number of parameters for each command. These generate data are then stored within the *DynamoDB* database, also hosted by *Amazon*. This database essentially creates a log of all the commands (coded in a specific manner by the user) launched within a given interactive session. By accessing this log, a user application can then programmatically react to each command.

**Java Server Application**

The server application is meant to control the access to a repository of files containing relevant code fragments and error information for the supported programming languages. These files contain a selection of common programming constructs for each supported language, as well as sets of hints to help users solve specific errors encountered during the compilation process. Access to the code fragments and error information data is provided through a set of web services, making it possible for experienced users or system administrators to modify existing data and even add support for new languages after the system's deployment.

**Client Web Application**

The client is a web application that provides the user with the visual feedback for the actions resulting from the spoken commands. This is essentially a prototype programming environment that integrates the voice command processing capabilities of the *Alexa* assistant. This results in most of its functionality being directly controllable through voice commands.

Voice control of the application is achieved by having it retrieve the commands registered within the *DynamoDB* database and applying a series of processing and interpreting steps. These read the name and parameters of the commands logged within the *DynamoDB* database and generate user-specified behaviors. In our case, these behaviors are the actions required to control the programming environment.

Accessing the data logged within the *DynamoDB* database can only be achieved through the *Amazon API Gateway* and *AWS Lambda Function Backend*, which are the mechanisms put in place by Amazon to enforce access control and security.

The graphical user interface of the client application consists of three major areas which can be seen in Figure 3. The first one (left side) contains a code editor where the code is inserted after the relevant commands are processed. A user can select a programming language from the top of the page, and also select a theme from one of the editor themes, located at the bottom of the area. The second area (top-right) provides a log that lists the virtual assistant's feedback. The feedback is chronologically presented so the user can see the order of the added instructions and their results. In the last area (bottom-right) there is the output of the interpreter/compiler. This area contains the result that can either be an error or the expected output of the executed code.

**Grammar of interaction model**

The grammar of the commands used consists of nine main elements. These can be grouped into two categories: code management and application management. Those from the first category are related to creating programs, variables or operators, navigating through the code, inserting code snippet such as "if", "while", "bubble sort", and printing text. Those from the second category are related to: application start, helping with errors, running programs, wait actions and setting the feedback length/complexity.

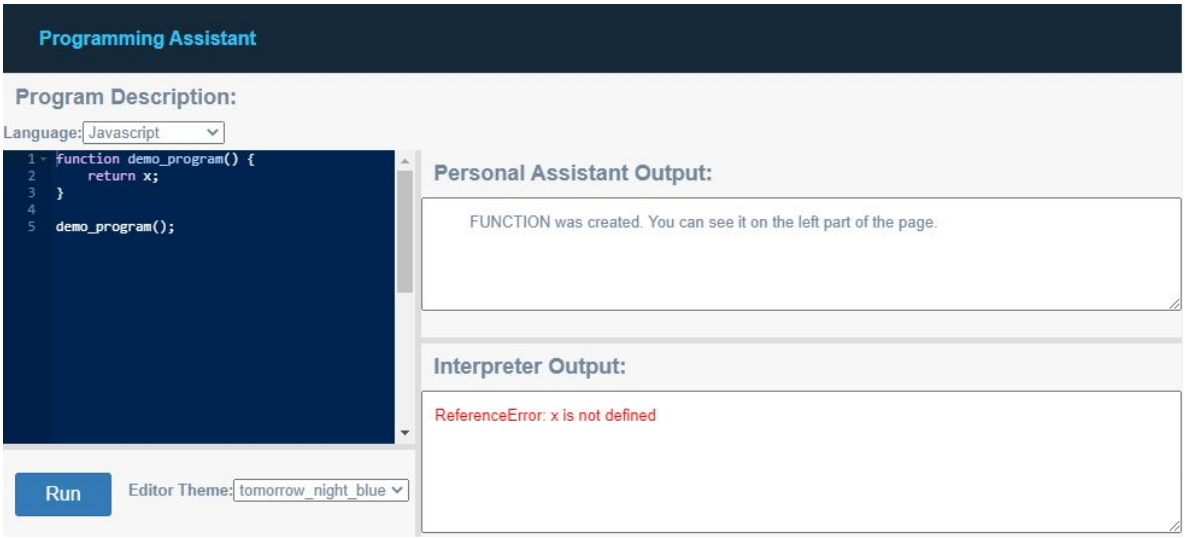


**Figure 3. User interface of the integrated programming environment**

The *start* command, is used to activate the application. The instruction responsible for managing the errors that appear after running the program is *help error*. The *run* command is used in order to start running the code and obtain a result from de compiler. *Wait*, used for increasing the time period in which *Alexa* is active, can be said when the user needs more time and doesn't want the skill to stop. The *set_reponse* command is used when the user wants to select the type (length) of voice feedback to be received from *Alexa*. The latter can be long – *Alexa* details the action, short – *Alexa* says only that it intercepts the command, and no – *Alexa* doesn't give any feedback.

The *create* command is used to add a new program template which has a specific name. Also, with this one, the user can add a new named variable or an arithmetic operator like *plus*, *minus*, *multiply* or *divide*. *Go to* permits navigating to different lines of code or getting into different named functions. *Insert* lets the user add templates for different programming statements like "if", "switch", "for" and "while". Also, using this command, users can add sorting algorithms. In order to print a sentence on the screen, the user need to say *write* followed by what he wants to print.

The grammar is a simple one but offers the most common commands in terms of a programming language. Simplicity is an essential feature of the dialogue model in such a situation, for a novice user. The user can express himself briefly and concisely through a well-defined set of keywords. The grammar has several levels of expansion. The rules consist of a terminal (uppercase) and a non-terminal (lowercase). The terminals have synonyms, so the user has multiple way to interact with the system without being constraint to use a strictly set of words.

```
command ::= start | create | go_to | insert | HELP ERROR
        | write | RUN | WAIT | set_response
create ::=CREATE OPERATOR operator_name
        |CREATE  PROGRAM program_name
        |CREATE  VARIABLE variable_name
go_to ::= GO TO LINE number
        | GO TO FUNCTION program_name
insert ::= INSERT structure | INSERT FOR number
structure ::= IF | SWITCH | WHILE | DO WHILE
        | BUBBLE SORT | MERGE SORT
set_reponse ::= SET RESPONSE response
response ::= NO | SHORT | LONG
```

## SYSTEM EVALUATION

This section presents an evaluation of the system's functionality and performance. The tests were designed to verify functional requirements of the system, while also validating the interaction between the system's components.

### Performance Evaluation

The performance evaluation of the system was done by analyzing its average response time for user requests. In this context, a user request refers to the *HTTP* request submitted from the user interface (through Alexa and the associated command interpretation logic) to the two application servers: the command history server, deployed on *Amazon's* infrastructure, and the server hosting the language construct and error data, hosted on the local machine. To evaluate their performance, a few endpoints were chosen to measure the time required for a response to be received. To get accurate results and level out any irregularities, each server was subjected to five calls for the same endpoint, and the response times were averaged. The results obtained can be seen in Figure 4.

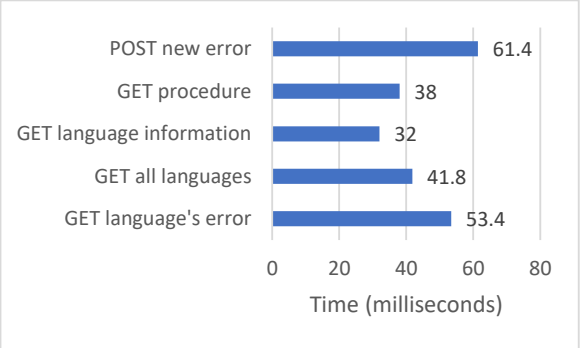


**Figure 4. Response time of local server**

Referring to Figure 4, which presents the performance evaluation of the local server responsible for delivering language syntax and error data, the first test was performed to add a new error interpretation message and the call requires the longest amount of time to complete (61.4 ms). This result is to be expected, as the request relies on a POST method that contains an object with 4 parameters in the body of the message, so it requires more processing time. The following methods are of the GET type. The second call was performed in order to measure the time required to extract detailed information on specific errors. The average response time was 38 ms. Analogous to the second test, the third was created to request the data in the form of syntactic constructs for a given programming language, resulting in a similar processing time of 32 ms. The call to fetch the syntax constructs data from all the registered languages within the database takes an average of 41.8 ms, and is reasonable because the result is in the form of a list of four elements of a complex data type. The last test, aimed at measuring the time required to fetch the

sets of errors associated with a language, averages a response time of 53.4 ms.

Figure 5 shows the average response times for the *NodeJS* server developed using *AWS Lambda* and hosted on *Amazon's* infrastructure. The first test was performed to verify the time required to insert an error into the NoSQL *ErrorLearnProgramming* database, with a resulting average time of 133.6 ms. The second test consists of deleting the entire database that contains the user's history of programming errors. As a sequential deletion was required, a longer response time (286 ms) was expected. The last test shows the performance of the server in terms of reading databases. This reading is an action that is performed very often, since the system polls the database to monitor changes in the command history. The average access time is 266.2 ms which is a reasonable time considering that oftentimes, the response consists of an array of approximately 30 complex object elements.

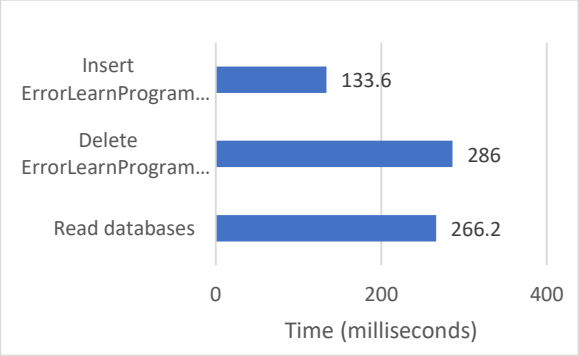


**Figure 5. Response time of AWS Lambda**

### Usability Evaluation

The ability of the system to behave as expected, to avoid unwanted command errors or to lose control is assessed below. In this analysis, the fulfillment of some use case scenarios was monitored, in an increasing order of complexity.

The tests included a variable number of instructions to analyze the dependence between the complexity of the scenario and the number of errors occurring. By error, in this context, we mean a situation in which *Alexa* did not understand the command and/or failed to act appropriately. Three types of scenarios were tested: low complexity (5 commands), medium complexity (15 commands) and high complexity (25 commands). These scenarios do not necessarily involve completely distinct command, but lead to distinct outcomes.

Figure 6 shows the influence of the number of commands on the number of errors. When referring to errors, it must be understood that they are errors of control of the system or errors of misunderstanding of the spoken words. In the case of the scenario containing 15 instructions, only one error occurred. This was related to semantics, more precisely the sentence was not understood by the vocal assistant. As for the last scenario consisting of 25 commands, the two errors occurred, also at the semantic level.

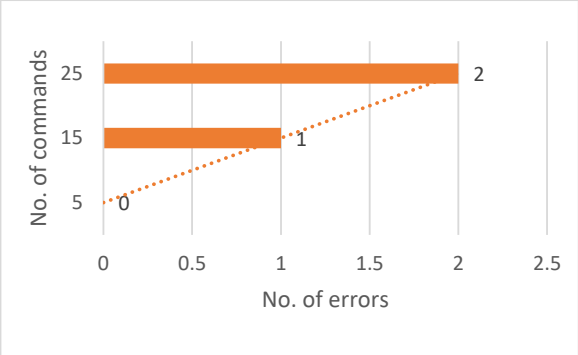


**Figure 6. Commands - errors dependency**

Figure 7 shows the relationship between the number of commands and the time required to fulfill them. As can be observed, the total time required for the completion of a scenario increases linearly, while the average time for realizing a command marginally fluctuates around the 10 seconds value. These measured time intervals include both the time required for the utterance of the command and that necessary for *Alexa's* spoken feedback. It should be noted, however, that we instructed *Alexa,* at the beginning of the test, to provide only the strictly necessary information. Therefore, the feedback messages were as short as possible.

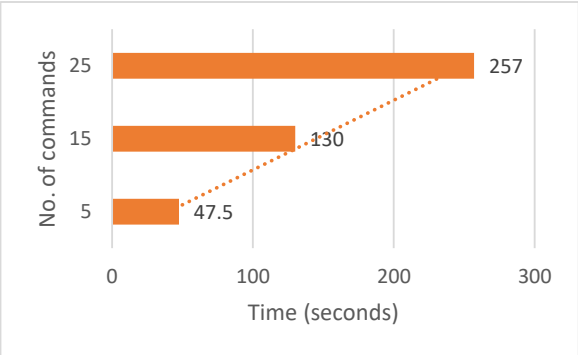


**Figure 7. Commands - time dependency**

#### *Heuristic Evaluation*

Heuristic evaluation has been defined as a usability engineering method that aims to identify usability issues in the design of a user interface.

An initial set of nine heuristics is given by Molich and Nielsen in 1990. After that, Nielsen in 1994, refined these heuristics and proposed a set of 10 usability principles [16]. This section contains an analysis of the system from the point of view of Nielsen's ten heuristics.

**Visibility of system status** - Information about the system's status and command execution is provided to the user in a reasonable amount of time. The mean observed time for the execution of a command is 10 seconds, from the utterance of the command to the voice feedback received from Alexa. The system can provide redundant, text-based feedback, containing additional information. Both types of feedback are supplied at the same time – immediately after the execution of the command. There is also the visual feedback, observed through the changes affected by the commands regarding cursor movement, code insertion, code folding and so on.

**Match between system and the real world** - The system uses terms that a novice user in the field knows, uses simple words and familiar concepts providing explanations where needed. This heuristic is applied in terms of verbal dialogue between the user and *Amazon Echo Dot*, but also in the web application by providing information in an easy to understand form.

**User control and freedom** – This is accomplished through several means: the system gives the user the opportunity to choose how he wants to be given the feedback, offers the possibility to do undo and redo commands and can help the user navigate through and solve errors. Another feature is the ability to extend the time Alexa listens to the user by using the wait command, or stop the interaction entirely by employing a built-in command.

**Consistency and standards** – The consistency of the system is respected both at the level of voice commands by employing the same keyword to address similar operations, minimizing the size of the keyword vocabulary and at the textual feedback level, following a standardized notation and structure for both the information and error messages.

**Error prevention** - In this regard, the system does not provide much support, being susceptible to command interpretation errors. However, it provides visual feedback when writing a code sequence that does not follow the language syntax and grammar rules. This is displayed within the programming environment.

**Recognition rather than recall** - This heuristic is quite difficult to apply within a voice-based interaction model. However, there is little support from the system, reminding the user to select the type/length of feedback to be delivered, as well as select the desired programming language to work with. These prompts are delivered by the system at the start of a working session, thus relieving the user from the need to remember the format of these particular commands.

**Flexibility and efficiency of use** - The accelerators made available by the system that make an expert user more efficient are the following: the option of not having a voice response (thus minimizing overall interaction time) and the possibility of uttering several commands at a time joined together by the particle *and*. As of this moment, up to two joined commands have been consistently observed to be executed correctly by the system.

**Aesthetic and minimalist design** - The dialogue required to achieve the desired effects is minimal without the need to provide non-essential information. The system does not ask the user for irrelevant or unnecessary information. Also, the visual interface of the programming environment has been kept as simple and clean as possible – as seen in Figure 3.

**Help users recognize, diagnose, and recover from errors** - System error messages are presented in textual form, without using encodings. Furthermore, the system is capable of providing the users with hints associated to each generated error. These hints are user-customizable and can thus be modified, without altering the system's implementation.

**Help and documentation** – At the moment, extensive documentation for the system is not available, as it is an ongoing work. However, provisions can be made to integrate a documentation within the webpage hosting the development environment and future developments can see the use of *Alexa* to navigate and find items of interest based on voice commands.

## CONCLUSION

The system described within this paper aims to take advantage of the intuitive and easy to use voice-based interaction model in order to facilitate the teaching of basic programming concept to novice users. To this end, it employs Amazon Alexa's API for creating custom voice commands (also referred to as skills). The proposed command set is limited in size and emphasizes language simplicity and compactness.

The contributions of the implemented project include the integration of an existing intelligent voice controlled assistant within a programming environment, the design and implementation of a set of simple and concise commands to manage the interaction between the user and the system and the development of a scalable support infrastructure allowing for the addition of multiple programming languages.

Thorough this paper, the basic architecture, components and functionality of the system are presented along with an evaluation of its performance – in terms of server response time – and usability. The latter is expressed in terms of the system's average command execution time, errors per command ratio and also by considering Nielsen's ten usability heuristics.

## REFFERENCES

1. Kumar, R., "Human Computer Interaction", Laxmi Publications, (2008), ISBN: 978-8131802809
2. Hoy, M., "Alexa, Siri, Cortana, and More: An Introduction to Voice Assistants," Medical Reference

Services Quarterly, vol. 37, pp. 81-88, (2018), doi: 10.1080/02763869.2018.1404391

3. Miluț, C., Iftene, A. and Gîfu, D., "Iasi City Explorer - Alexa, what can we do today?" in Proceedings of RoCHI - International Conference on Human-Computer Interaction, pp. 139-164, (2019), ISSN 2501-9422
4. Lopatovska, I., "Overview of the Intelligent Personal Assistants", Ukrainian Journal on Library and Information Science, pp. 72-79, (2019), doi: 10.31866/2616-7654.3.2019.169669
5. Google, "Google Assistant, your own personal Google", (2020), https://assistant.google.com/
6. Hadi, M. S., Shidiqi, A. A. and Zaeni, I. A. E., "Voice-Based Monitoring and Control System of Electronic Appliance Using Dialog Flow API Via Google Assistant," 2019 International Conference on Electrical, Electronics and Information Engineering (ICEEIE), Denpasar, Bali, Indonesia, pp. 106-110, (2019), doi: 10.1109/ICEEIE47180.2019.8981415.
7. Matei, A., and Iftene, A., "Smart Home Automation through Voice Interaction," in Proceedings of RoCHI - International Conference on Human-Computer Interaction, pp.132-137, (2019), ISSN 2501-9422
8. Apple, "Siri - Apple,", (2020), https://www.apple.com/siri/
9. Aron, J., "How innovative is Apple's new voice assistant, Siri?", New Scientist - NEW SCI, vol. 212, pp 24-24, (2011), 10.1016/S0262-4079(11)62647-X
10. Microsoft, "What is Cortana?", (2020), https://support.microsoft.com/ro-ro/help/17214/cortana-what-is
11. Godinez, J. E. and Jamil, H., "Meet Cyrus: The Query by Voice Mobile Assistant for the Tutoring and Formative Assessment of SQL Learners", SAC '19: Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, pp. 2461-2468, (2019), ISBN: 978-1-4503-5933-7, doi: 10.1145/3297280.3297523
12. Dickinson, J., "Amazon.com: Voice Coder: Alexa Skill", (2020), https://www.amazon.com/Jimmy-Dickinson-Voice-Coder/dp/B07HFWQPKN
13. Fireberger, A., "Amazon.com: Code: Alexa Skill", (2020), https://www.amazon.com/aviram-fireberger-Coder/dp/B07P81FZVL
14. Manish, A., "Amazon.com: C Programming Quiz: Alexa Skill", (2020), https://www.amazon.com/Manish-A-C-Programming-Quiz/dp/B07Z514B3F
15. Dephony, "Amazon.com :CS Guru: Alexa Skill", (2020), https://www.amazon.com/DEPHONY-CS-Guru/dp/B07VRF5BKR
16. Pribeanu, C., "Tendinte actuale în evaluarea interfetelor om-calculator," Informatica Economica, vol. 2 nr. 4(8), pp. 21-25, (1998), ISSN 1453-1305